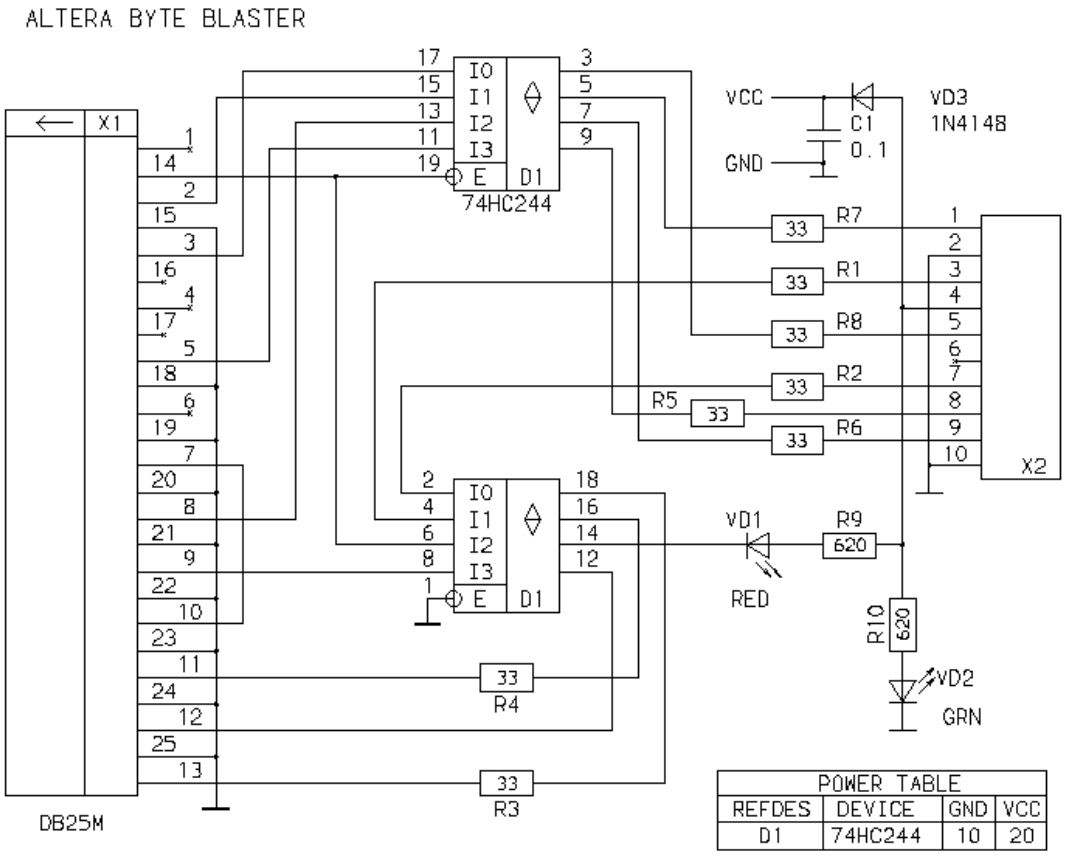
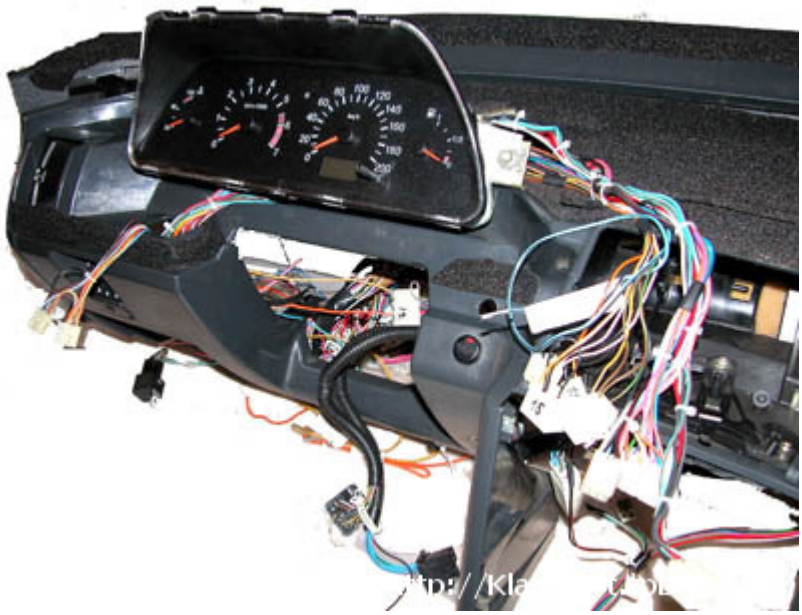
# Шины

Перед тем, как познакомиться с регистрами, давайте рассмотрим некоторые правила рисования схем. Возьмём первую попавшуюся схему из Интернета. Не важно, что она делает, но давайте представим себе, что мы хотим в ней разобраться.



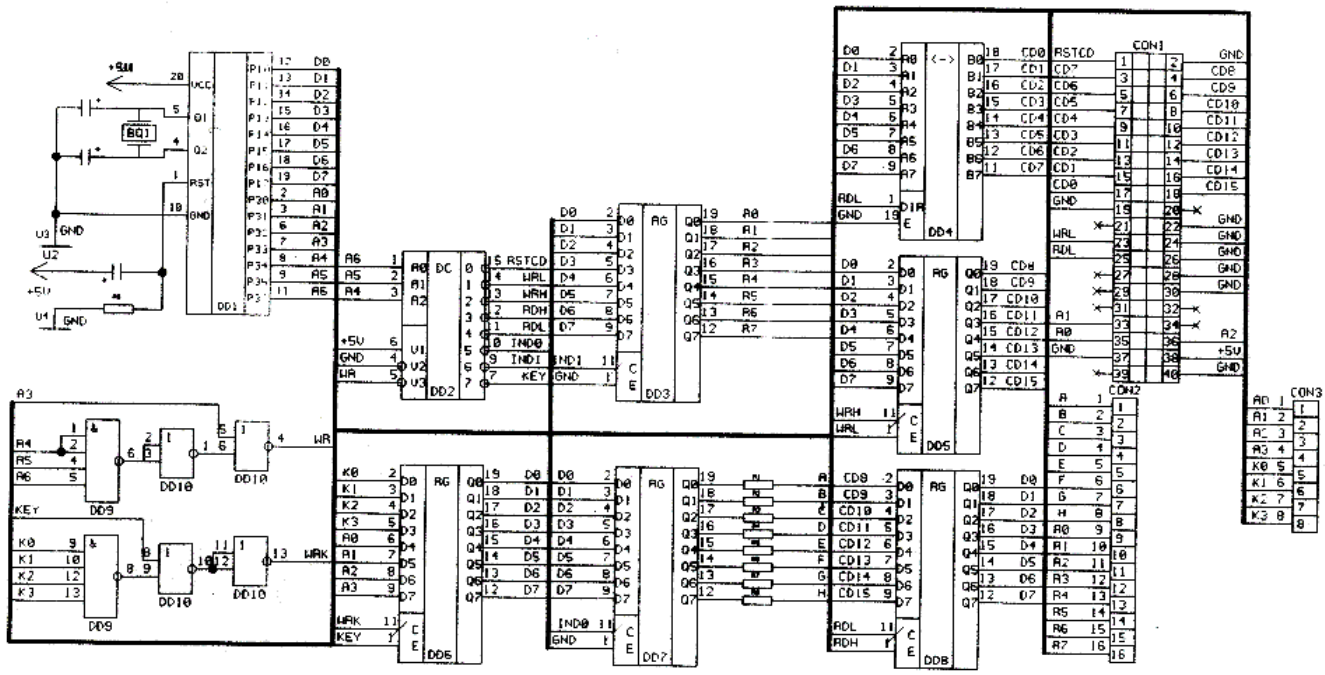

Скажем просто: На этой схеме 4 провода идут в одну сторону, 3 – в обратную, и один провод – закручивается назад. Попробуйте-ка проследить любой из сигналов! И это – всего восемь проводов. А если их будет больше? А если надо будет протащить 32 провода данных (компьютеры же у нас 32 разрядные) между десятью местами? У нас схема превратится в нагромождение верёвок, среди которых будут отдельные вкрапления логических элементов (ну прямо как некоторые ТВ-каналы – это нагромождение рекламы, среди которой иногда появляются вкрапления программ). Пример такой схемы – навесное оборудование вертолёта в проекте PNRS (АстроСофт к этой части не имеет никакого отношения). Авторы схемы, если им что-то нужно, начинают вести пальцем по интересующей связи, теряют, ведут отвёрткой (она тоньше), теряют, плюют, берут маркер, отрисовывают интересующую связь определённым цветом, успокаиваются… До поиска следующей связи, либо взятия новой схемы. Но нам же этого не надо. Нам нужно, чтобы схема была ЛЕГКО читаемой…

А давайте скрутим все провода в один пучок. У нас получится толстый жгут, от которого иногда отходят отдельные проводочки, на концах которых имеются пометки. Такие жгуты были весьма популярны в своё время, так что это – не гипотетическая вещь, а вполне реальная. Будучи не дома, я не смог сфотографировать что-нибудь реальное. Порывшись в поисковиках, нашёл в качестве самой симпатичной иллюстрации вот эту:



В общем, нам не интересно, как именно провод идёт по пучку, нам важно, что в нужном нам месте провод вынырнет из пучка, и у него будет маркировка, однозначно идентифицирующая провод.

При рисовании схем пошли тем же путём. По схеме кладётся широкая линия, которую называют шиной. Где необходимо, там от шины отходят тонкие провода с персональными именами. Будучи стеснён в наличии собственных схем (они все лежат за тысячи километров от меня, дома), я слазал в Интернет и нашёл первую попавшуюся сему, иллюстрирующую этот подход



Цифра около вывода микросхемы – это номер ножки, на которую выведен контакт логического элемента. К счастью, при использовании ПЛИС об этом можно забыть, как о страшном сне, так что я просто написал об этом, но расписывать подробно не стану. Просто не пугайтесь. А надпись около широкой линии – это как раз наименование сигнала в шине. Грубо говоря, это метка на проводе, куда он тянется.

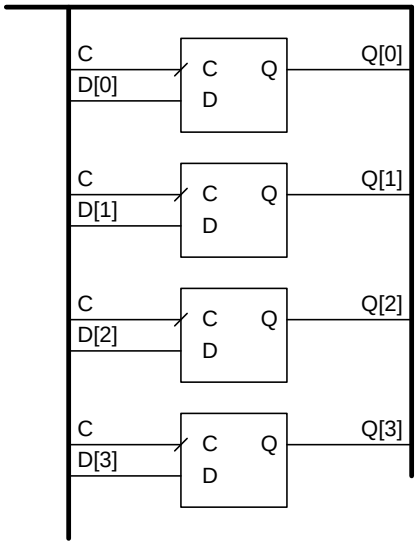
При работе в редакторах, найти одноимённые провода можно поиском, а на бумаге – при определённом уровне тренировки, на глазок. Тренироваться долго не надо, проверено.

Этот формат долгое время был стандартом. Однако, в последнее время широкую линию рисовать перестали. Просто обрывают линию и пишут её название. Это позволяет ещё более упростить схему. Собственно, я уже приводил достаточно вольные примеры, где около стрелки было указано, куда она ведёт. Вот это как раз пример того стиля.

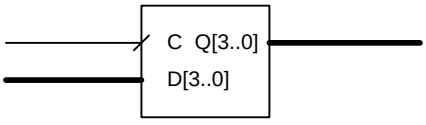
# Параллельные регистры

## *Теория*

Ну, вот. Теперь, когда мы познакомились с шинным методом, можно поговорить и о параллельных регистрах. Собственно, после того, что описано выше, с ними всё просто. Давайте нарисуем 4 D-триггера в таком включении:




Что это такое? Это 4 триггера, которые защёлкнут в себе 4 бита данных одновременно по фронту сигнала C. Так вот. Такое включение называется параллельным регистром. И по этому случаю, для него даже придумали особое обозначение. Так как наши лекции плавно перетекут в рассуждения о ПЛИС, я оставлю в стороне классическое обозначение регистров, и сразу нарисую регистр с точки зрения редактора схем для ПЛИС (хотя бы потому, что оно проще).




Почему я взял разрядность 4? Да просто потому что картинка из четырёх триггеров выглядит компактней. А так – хоть 8, хоть 16, хоть 32 триггера… Получим 8, 16 и 32 разрядные регистры. И с точки зрения программиста, это будут переменные типа BYTE, WORD и DWORD соответственно, ведь каждый триггер в регистре защёлкивает один бит данных. Вот так легко и непринуждённо мы научились вводить в наши схемы переменные для запоминания тех или иных полезных вещей.

Маленькое лирическое отступление. Вообще, это, конечно, лучше повертеть в руках, но попробую описать словами. Мы все говорим «Процессор», хотя во всех книгах пишут «Микропроцессор». Почему микро? Так уж получилось, что мне довелось потрогать руками процессор ЭВМ СМ1420. Он по размеру был с пару ящиков комода. И если в справочнике на микропроцессор на картинке со структурной схемой есть блок регистров общего назначения (для IBM PC это EAX, EBC, ECX и т.п., для PDP-11 это R0, R1, R2…), то в процессоре есть плата регистров. Размером с ноутбук (если не больше, давно не видел). И каждый регистр был выполнен в виде двух микросхем К589ИР12, которые представляют собой ни что иное, как описанные выше параллельные регистры. Ощущение, когда трогаешь пальцем регистр R2, причём понимаешь, что это не R0, не R1 и не R3 – не передаваемые. А впоследствии всё просто ушло из разных микросхем в одну…

Итак. Параллельный регистр – это чрезвычайно важная в цифровой технике вещь, так как она хранит цифровые данные, но на самом деле, это не более, чем много D-триггеров, у которых входы C объединены друг с другом… Вот так.

## *Практика*

Ну что же, давайте сделаем некое подобие порта LPT. Но не полный, а только его фрагмент, который иллюстрирует работу регистра. Делать мы его будем для уже устаревшей шины ISA, но на ней проще всего всё иллюстрировать.

Как работает порт LPT? Мы пишем константу в порт 378 или 278 (LPT1 и LPT2), эта константа в двоичном виде появляется на контакиех 2-9 разъёма LPT и остаётся там до тех пор, пока мы не запишем иную константу. Вот и всё. Попробуем записать двоичную константу 10101010 (шестнадцатеричное значение AA)

На ассемблере это выглядит так:

```
mov    dx,278h
mov    al,0aah
out    dx,al
```

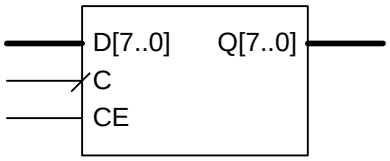
На Borland Pascal – так:

```
port[$278] := $aa;
```

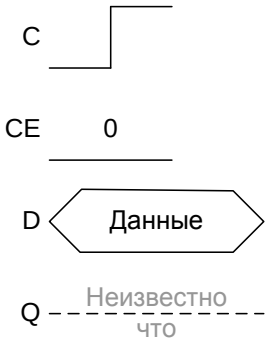
На C++ (не пытайтесь выполнить под Windows, на халяву не сработает, если надо – научу, как сделать, чтобы сработало):

```
_outp (0x278,0xaa);
```

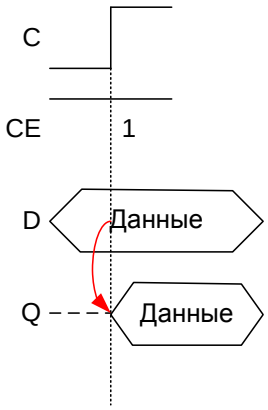
Ну вот и займёмся изготовлением такой полезной вещи, которая будет отображать на светодиодах двоичный код, посланный в порт 0x278 (базовый адрес LPT2, так как считаем, что LPT1 у нас занят). Для начала рассмторим новый тип регистра. У него кроме входов D и C, есть вход CE (Clock Enable).




Этот регистр защёлкнет данные не при любом фронте C, а только при таком, когда CE=1. То есть, при вот такой ситуации, данные не будут защёлкнуты, так как CE=0. На выходе останется то, что было раньше.



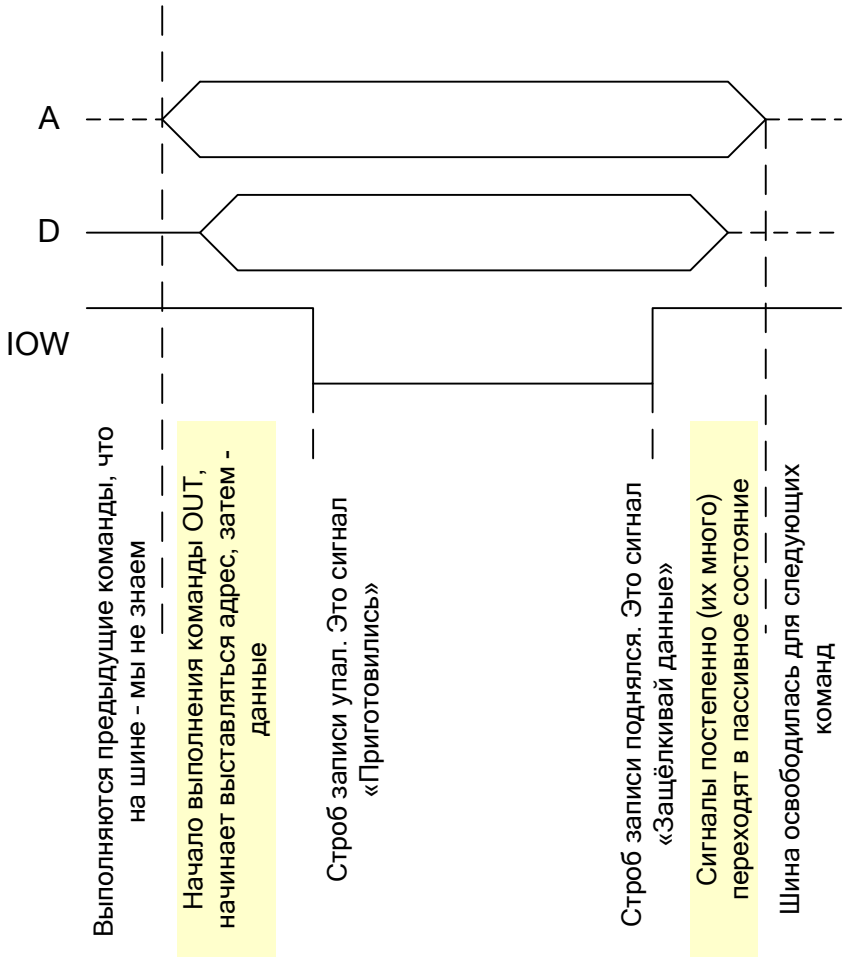

А вот при такой – данные, которые были на входах D, по фронту C перескочат в Q



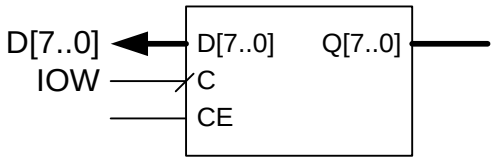

А теперь познакомимся с шиной ISA. На ней есть такие полезные для нашей задачи сигналы:

| **Сигнал** | **Назначение** |
|---|---|
| A[21..0] | Шина адреса. На неё передаётся адрес в двоичном виде. При работе с портами, используются только 12 младших линий (от 0 до 11). Остальные – просто игнорируем. |
| D[15..0] | Шина данных. На неё передаются данные в двоичном виде. |
| IOW | Строб записи. В пассивном состоянии равен 1. Когда процессор встречает ассемблерную команду OUT, он сначала выставляет адрес и данные, а затем – дёргает этот сигнал в 0, затем – в 1. |

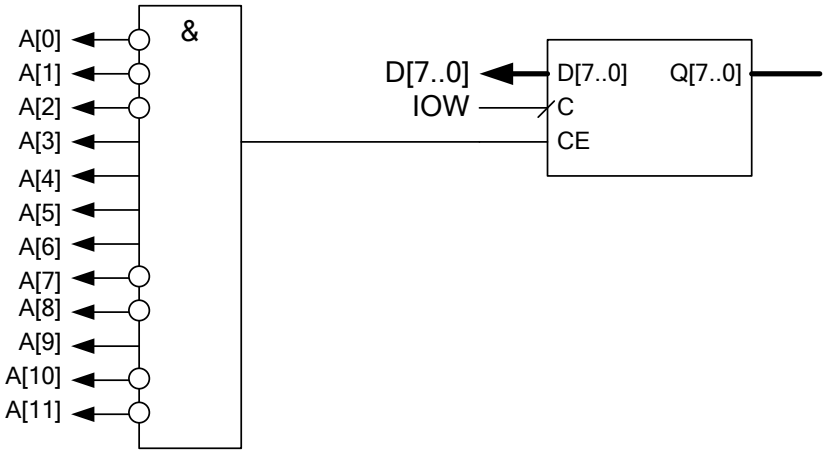
Итак. Когда мы выполнили команду ассемблера OUT, на шине ISA происходит следующее:



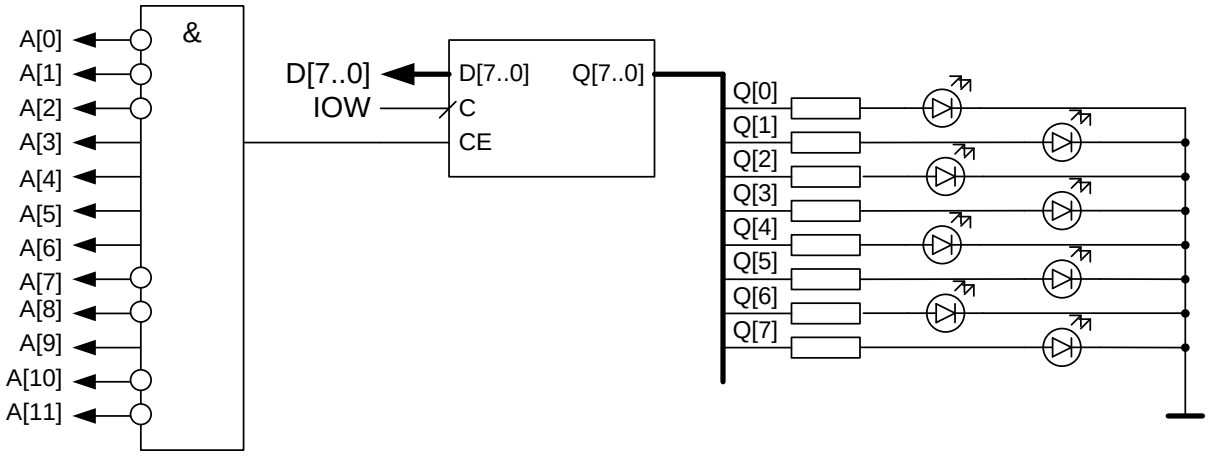

Ну что же. Если мы изучаем регистры, то наверное, на нашей схеме он обязан быть, это вполне предсказуемо. Кладём его на схему. Шину данных регистра подключаем к шине данных ISA, правда, нам не надо все 16 бит, нам хватит восьми (мы же с байтом работаем). На вход C, как логично предположить из приведённой выше диаграммы, подаём сигнал IOW, так как он прекрасно укладывается в идеологию работы регистра (было бы удивительно, если бы не укладывался, шину же проектировали под имеющиеся микросхемы, а имелись именно регистры).




Что делаем с сигналом CE? Нам надо, чтобы там была единица тогда и только тогда, когда на шине адреса константа 0x278. Берём калькулятор и выясняем, что в двоичном виде это 001001111000. При работе с портами достаточно анализировать столько разрядов (12 штук). Мы уже знаем, что подобные вещи делаются при помощи функции (НЕ)12И. Получаем схему:




Остался выход из регистра. Можно его развести на разъём и получить настоящий LPT (Q[0] на контакт 2, Q[1] – контакт 3, Q[2] – контакт 4 и т.д). А можно для красоты наставить светодиодов (не забывая ограничивать ток, чтобы нам транзисторы не сожгло в двухтактных выходах, ещё не забыли, что это такое?).




Собственно, всё. Теперь у нас есть устройство, которое может мигать лампочками согласно программе, написанной для IBM PC. Ну, или частичный порт LPT. Сложно? По-моему, нет. 10 лет назад, я читал описания шины ISA и был уверен, что не смогу повторить сей подвиг. Всё потому, что там шина была описана полностью. А потом нашёл книжку отечественных авторов, где она тоже была описана кусками, и понял, что на самом деле, всё просто. PCI чуть сложнее (дойдут руки – и её замучаем), а что касается USB, то применяя правильные микросхемы, её можно свести к ISA-подобной структуре. Но не всё сразу, перед микроконтроллерами нам надо покончить с триггерами, да заняться сложной логикой. Но размер текста получился такой большой, что финальный разговор о триггерах мы отложим на следующую статью.

P.S. А если вместо порта 278 использовать порт 0x080 (просто поменять дешифратор), то получим супер прибор для диагностики убитых материнких плат. Так называемый, POST тестер. Разумеется, он будет диагностировать только материнские платы со слотом ISA, но всё же. Вроде материнка не запускается, а воткнул тестер и выяснил, чем она больна. Так что вот. Потихоньку-помаленьку, мы уже запчасти для компьютеров начинаем делать. И разве это сложно? Глаза боятся, а руки делают…